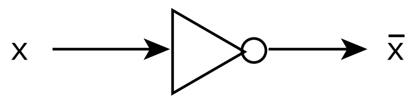
4. Logic Gates

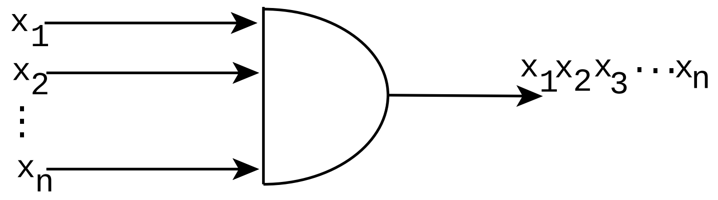
4.1. Logic Gates.

DEFINITION 4.1.1. *Boolean algebra can be used to model circuit design in an electronic device. Basic elements are* **gates**. *The three most common gates are*

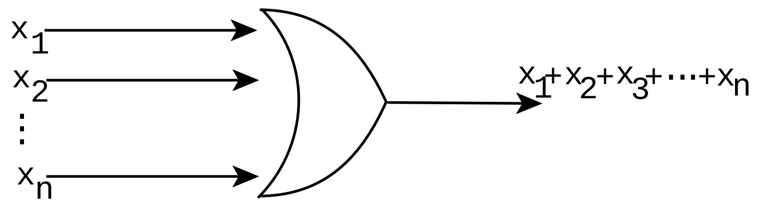
(1) Inverter:



(2) AND:



(3) OR:



Discussion

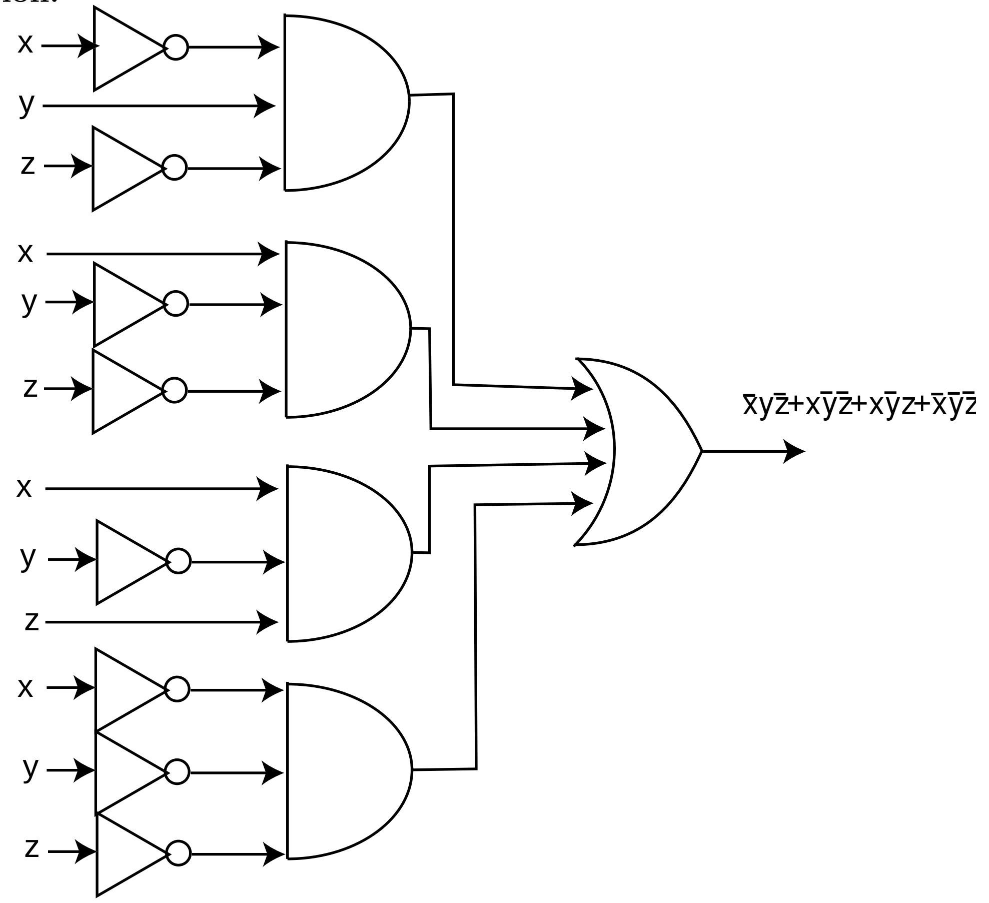
Gates are the basic building blocks of circuits. We combine gates to construct *combinatorial circuits* or *gating networks* that have as output a given Boolean expression.

4.2. Example 4.2.1.

EXAMPLE 4.2.1. *Construct a combinatorial circuit for the function F given by the following table:*

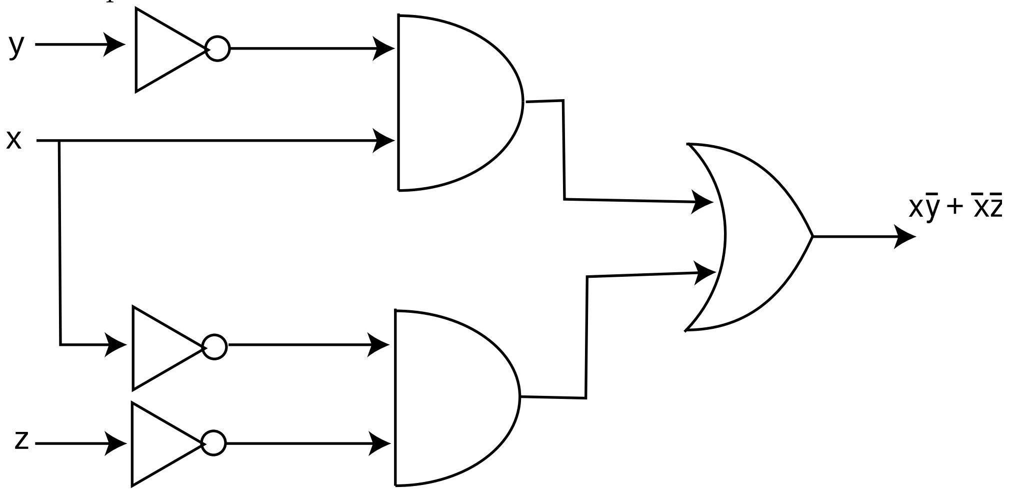
x	y	z	$F(x,y,z)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Solution.


x
y
z
x
y
z
x
y
z
x
y
z
$\bar{x}y\bar{z}+x\bar{y}\bar{z}+x\bar{y}z+\bar{x}\bar{y}\bar{z}$

Discussion

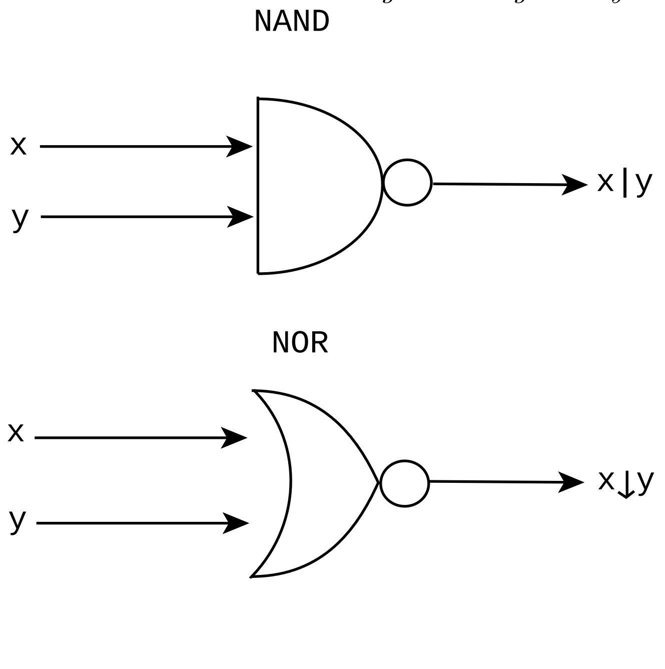
The figure given in the solution to example 4.2.1 comes from the disjunctive normal form for the function. The function is equivalent to $\overline{x}y\overline{z} + x\overline{y}\,\overline{z} + x\overline{y}z + \overline{x}\,\overline{y}\,\overline{z}$. This function is also equivalent to $x\overline{y} + \overline{x}\,\overline{z}$, so the combinatorial network below also has the same output.



Clearly the network given here is simpler than the one given in the solution for example 4.2.1. In this section we will not be concerned with simplifying the networks, only with getting the correct output using the gates discussed. Simplification processes will be addressed in the next set of lecture notes.

4.3. NOR and NAND gates.

Definition 4.3.1. *The NOR and NAND gates are given by*



Discussion

Recall the definition of NAND and NOR from *Representing Boolean Functions*. It was proven in that lecture that the sets $\{|\}$ and $\{\downarrow\}$ are both functionally complete. So a combinatorial circuit can always be created so that it consists only of NAND gates or only of NOR gates.

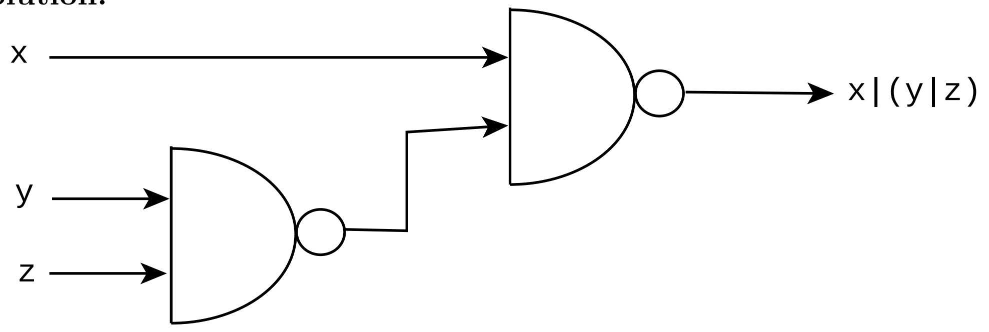
4.4. Example 4.4.1.

EXAMPLE 4.4.1. *Construct a circuit for the output of the expression*

$$\overline{x} + yz$$

using only NAND gates.

Solution.



Discussion

We use the following to get the expression in terms of NAND operators only.

$\overline{x} + yz = \overline{(x)\overline{(yz)}}$ by DeMorgan's Laws

$= (x)|\overline{(yz)}$ by the definition of NAND

$= x|(y|z)$ by the definition of NAND.

We then use this expression to make the circuit.

The expression used for the solution of Example 4.4.1 is not the only possible expression. We could have also found an expression for $\overline{x} + yz$ using the equivalences $\overline{x} = x|x$, $xy = (x|y)|(x|y)$, and $x + y = (x|x)|(y|y)$. Using these equivalences we get $\overline{x} + yz = (x|x) + (y|z)|(y|z) = \{(x|x)|(x|x)\}|\{[(y|z)|(y|z)]|[(y|z)|(y|z)]\}$. This expression is much more complex than the one used for the solution in Example 4.4.1, though!

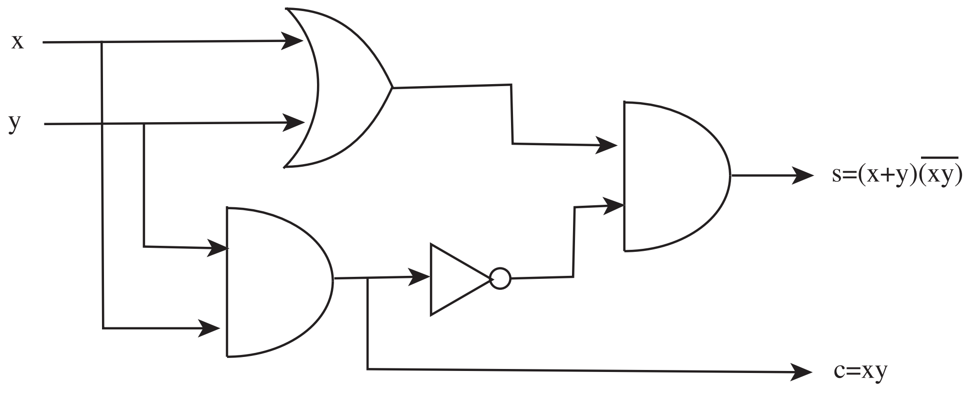
EXERCISE 4.4.1. *Construct a circuit for the output of the expression in Example 4.4.1 using only NOR gates.*

4.5. Half Adder.

Definition 4.5.1. *The* **half adder** *circuit has as input the variables x and y and has as output the variables s and c given by the table.*

Input		*Output*	
x	y	s	c
0	*0*	*0*	*0*
0	*1*	*1*	*0*
1	*0*	*1*	*0*
1	*1*	*0*	*1*

The functions $c = xy$ and $s = (x + y)\overline{(xy)}$ give the correct output for these functions. Therefore, the circuit for the half adder follows:

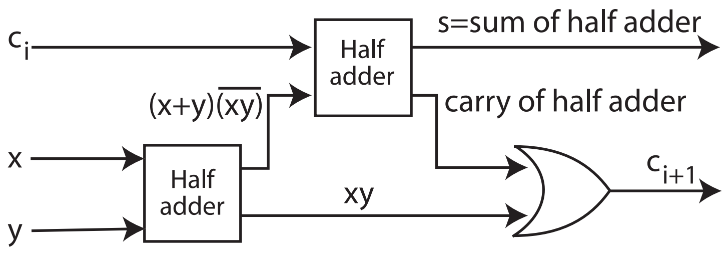


4.6. Full Adder.

Definition 4.6.1. *The* **full adder** *circuit has as input the variables x, y, and c_i and has as output the variables s and c_{i+1} given by the table.*

Input			*Output*	
x	y	c_i	s	c_{i+1}
0	*0*	*0*	*0*	*0*
0	*0*	*1*	*1*	*0*
0	*1*	*0*	*1*	*0*
0	*1*	*1*	*0*	*1*
1	*0*	*0*	*1*	*0*
1	*0*	*1*	*0*	*1*
1	*1*	*0*	*0*	*1*
1	*1*	*1*	*1*	*1*

We can use the half adder to calculate the full adder as shown in the circuit below.



Discussion

Recall from *Applications of Number Theory* the algorithm for adding two integers a and b written in base 2:

Let $a = (a_{n-1}a_{n-2}\cdots a_1a_0)_2$ and $b = (b_{n-1}b_{n-2}\cdots b_1b_0)_2$, and suppose the sum of a and b is $s = (s_ns_{n-1}s_{n-2}\cdots s_1s_0)_2$. Note that if the binary expansions for a and b do not have the same number of digits we add zeros to the left of the smaller one to make them the same length.

Since $s = a + b$ we have

$a_0 + b_0 = 2c_0 + s_0$

$a_1 + b_1 + c_0 = 2c_1 + s_1$

$a_2 + b_2 + c_1 = 2c_2 + s_2$

$$\vdots$$

$$a_{n-1} + b_{n-1} + c_{n-2} = 2c_{n-1} + s_{n-1}$$

$$s_{n-1} = c_{n-1}.$$

At the first stage of the addition we only add the first bits of a and b and get out the values for c_0 and s_0. The half adder gives us this operation. In fact, the half adder gives us a "first stage" of each of the following additions as well. However, we must go further than adding the bits of a and b to get the carries and sums on subsequent stages because we must also consider the carry from the previous addition. The full adder gives us the carry and sum when we input the appropriate bits for a and b and the previous carry.

Exercise 4.6.1. *Explain each stage of the algorithm used to find the sum of the binary numbers* 1011 *and* 110 *by giving each step including which adder is used and its input and output.*